KOTLIN

Język programowania dla Androida

Historia

Historia

- Kotlin został opracowany przez firmę JetBrains

Historia

- Kotlin został opracowany przez firmę JetBrains
- Prace rozpoczęto w 2011 r., od 2012 r. dostępny na licencji Apache 2.

Historia

- Kotlin został opracowany przez firmę JetBrains
- Prace rozpoczęto w 2011 r., od 2012 r. dostępny na licencji Apache 2.
- Nazwa pochodzi od wyspy koło Petersburga

Historia

- Kotlin został opracowany przez firmę JetBrains
- Prace rozpoczęto w 2011 r., od 2012 r. dostępny na licencji Apache 2.
- Nazwa pochodzi od wyspy koło Petersburga
- Kotlin 1.0 został udostępniony 15 lutego 2016 r.

Historia

- Kotlin został opracowany przez firmę JetBrains
- Prace rozpoczęto w 2011 r., od 2012 r. dostępny na licencji Apache 2.
- Nazwa pochodzi od wyspy koło Petersburga
- Kotlin 1.0 został udostępniony 15 lutego 2016 r.
- W pełni wspierany od Android Studio 3.0

Historia

- Kotlin został opracowany przez firmę JetBrains
- Prace rozpoczęto w 2011 r., od 2012 r. dostępny na licencji Apache 2.
- Nazwa pochodzi od wyspy koło Petersburga
- Kotlin 1.0 został udostępniony 15 lutego 2016 r.
- W pełni wspierany od Android Studio 3.0
- Aktualnie wersja 1.3

Historia

- Kotlin został opracowany przez firmę JetBrains
- Prace rozpoczęto w 2011 r., od 2012 r. dostępny na licencji Apache 2.
- Nazwa pochodzi od wyspy koło Petersburga
- Kotlin 1.0 został udostępniony 15 lutego 2016 r.
- W pełni wspierany od Android Studio 3.0
- Aktualnie wersja 1.3
- Od 7 maja 2019 Google uznał Kotlin za preferowany język dla Androida

Historia

- Kotlin został opracowany przez firmę JetBrains
- Prace rozpoczęto w 2011 r., od 2012 r. dostępny na licencji Apache 2.
- Nazwa pochodzi od wyspy koło Petersburga
- Kotlin 1.0 został udostępniony 15 lutego 2016 r.
- W pełni wspierany od Android Studio 3.0
- Aktualnie wersja 1.3
- Od 7 maja 2019 Google uznał Kotlin za preferowany język dla Androida
- Strona www.kotlinlang.org

Cechy

Cechy

- Mimo innej składni może współpracować z kodem w Javie

Cechy

- Mimo innej składni może współpracować z kodem w Javie
- Ma się kompilować równie szybko jak Java

Cechy

- Mimo innej składni może współpracować z kodem w Javie
- Ma się kompilować równie szybko jak Java
- Może się kompilować do bytecode'u (Java 6 i 8) jak i do Java Script'u i LLVM.

Cechy

- Mimo innej składni może współpracować z kodem w Javie
- Ma się kompilować równie szybko jak Java
- Może się kompilować do bytecode'u (Java 6 i 8) jak i do Java Script'u i LLVM.
- Android Studio wspiera konwersje kodu z Javy na Kotlin

Playground

Playground

- Twórcy języka oferują interaktywną piaskownicę, w której można się bawić fragmentami kodu pod adresem:

http://try.kotl.in

- Twó ę,
  w któ d
  adres

Kotlin

LEARN COMMUNITY TRY ONLINE

Shortcuts | Convert from Java | Fullscreen

Examples > Hello, world! > Simplest version > Simplest version.kt

- Examples
  - Hello, world!
    - Simplest version
      - Simplest version.kt
    - Reading a name from the...
    - Reading many names fro...
    - A multi-language Hello
    - An object-oriented Hello
  - Basic syntax walk-through
  - Destructuring declarations and
  - Delegated properties
  - Callable references
  - Longer examples
  - Problems
  - Canvas
- Kotlin Koans 0/42
- Kotlin in Action
- Advent of Code (log in)
- My programs (log in)

Save | Save as | Arguments | JVM | Run

Program arguments

```
/**
 * We declare a package-level function main which returns Unit and takes
 * an Array of strings as a parameter. Note that semicolons are optional.
 */

fun main(args: Array<String>) {
    println("Hello, world!")
}
```

Compilation completed successfully | On-the-fly type checking

```
Hello, world!
```

Problems view | Console | Generated classfiles

This demo is running on Kotlin v. 1.2.21

Cechy

Cechy

- Podobnie jak w Swift średniki na końcu linii są opcjonalne

Cechy

- Podobnie jak w Swift średniki na końcu linii są opcjonalne
- Kotlin wykorzystuje inferencję typów

Cechy

- Podobnie jak w Swift średniki na końcu linii są opcjonalne
- Kotlin wykorzystuje inferencję typów
- Podział zmiennych na modyfikowalne **var** (mutable) i stałe **val** (immutable)

Typy podstawowe

Typy podstawowe

- Byte
- Short
- Int
- Long
- Float
- Double

Typy podstawowe

- Byte
- Short
- Int
- Long
- Float
- Double
- Boolean
- Char
- String

Przykłady deklaracji zmiennych

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

```
val maxUserCount = 20
```

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

```
val maxUserCount = 20
```

Deklaracja stałej Int przez inferencję

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

```
val maxUserCount = 20
```

Deklaracja stałej Int przez inferencję

```
val myString = "The quick brown fox"
val uppercase = myString.toUpperCase()
```

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

```
val maxUserCount = 20
```

Deklaracja stałej Int przez inferencję

```
val myString = "The quick brown fox"
val uppercase = myString.toUpperCase()
```

Deklaracja zmiennej String przez inferencję
Wywołanie metody

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

```
val maxUserCount = 20
```

Deklaracja stałej Int przez inferencję

```
val myString = "The quick brown fox"
val uppercase = myString.toUpperCase()
```

Deklaracja zmiennej String przez inferencję
Wywołanie metody

```
val length = myString.length
val result = myString.contains("fox")
```

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

```
val maxUserCount = 20
```

Deklaracja stałej Int przez inferencję

```
val myString = "The quick brown fox"
val uppercase = myString.toUpperCase()
```

Deklaracja zmiennej String przez inferencję
Wywołanie metody

```
val length = myString.length
val result = myString.contains("fox")
```

Wywołania metod

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

```
val maxUserCount = 20
```

Deklaracja stałej Int przez inferencję

```
val myString = "The quick brown fox"
val uppercase = myString.toUpperCase()
```

Deklaracja zmiennej String przez inferencję
Wywołanie metody

```
val length = myString.length
val result = myString.contains("fox")
```

Wywołania metod

```
val myInt = 10
val myFloat = myInt.toFloat()
```

Przykłady deklaracji zmiennych

```
var userCount = 10
```

Deklaracja zmiennej Int przez inferencję

```
var userCount1: Int
userCount1=10
```

Deklaracja zmiennej Int wprost

```
val maxUserCount = 20
```

Deklaracja stałej Int przez inferencję

```
val myString = "The quick brown fox"
val uppercase = myString.toUpperCase()
```

Deklaracja zmiennej String przez inferencję
Wywołanie metody

```
val length = myString.length
val result = myString.contains("fox")
```

Wywołania metod

```
val myInt = 10
val myFloat = myInt.toFloat()
```

Konwersja na Float

Typy nullowe

Typy nullowe

- Domyślnie w Kotlinie zmienna nie może mieć przypisanej wartości null

Typy nullowe

- Domyślnie w Kotlinie zmienna nie może mieć przypisanej wartości null

```
var unknown:String = null
```

Typy nullowe

- Domyślnie w Kotlinie zmienna nie może mieć przypisanej wartości null

```
var unknown:String = null
```

```
Simplest version.kt
Error:(8, 22) Null can not be a value of a non-null type String
```

Typy nullowe

- Domyślnie w Kotlinie zmienna nie może mieć przypisanej wartości null

```
var unknown:String = null
```

```
Simplest version.kt
  Error:(8, 22) Null can not be a value of a non-null type String
```

- Próba odwołania się do zmiennej niezainicjalizowanej powoduje błąd

Typy nullowe

- Domyślnie w Kotlinie zmienna nie może mieć przypisanej wartości null

```
var unknown:String = null
```

```
Simplest version.kt
Error:(8, 22) Null can not be a value of a non-null type String
```

- Próba odwołania się do zmiennej niezainicjalizowanej powoduje błąd

```
8   val unknown:String
9   println(unknown)
10  unknown="Test"
11  println(unknown)
```

Typy nullowe

- Domyślnie w Kotlinie zmienna nie może mieć przypisanej wartości null

```
var unknown:String = null
```

```
Simplest version.kt
Error:(8, 22) Null can not be a value of a non-null type String
```

- Próba odwołania się do zmiennej niezainicjalizowanej powoduje błąd

```
8   val unknown:String
9   println(unknown)
10  unknown="Test"
11  println(unknown)
```

```
Simplest version.kt
Error:(9, 12) Variable 'unknown' must be initialized
```

Typy nullowe

Typy nullowe

- Dodając do nazwy typu ? otrzymujemy zmienne, które mogą mieć wartość null

Typy nullowe

- Dodając do nazwy typu ? otrzymujemy zmienne, które mogą mieć wartość null

```
var unknown:String? = null
println(unknown)
unknown="Test"
println(unknown)
```

Typy nullowe

- Dodając do nazwy typu ? otrzymujemy zmienne, które mogą mieć wartość null

```
var unknown:String? = null
println(unknown)
unknown="Test"
println(unknown)
```

```
null
Test
```

Typy nullowe

- Dodając do nazwy typu ? otrzymujemy zmienne, które mogą mieć wartość null

```
var unknown:String? = null
println(unknown)
unknown="Test"
println(unknown)
```

```
null
Test
```

- Zmiennych typu nullowego nie można wprost podstawiać pod zwykłe zmienne

Typy nullowe

- Dodając do nazwy typu ? otrzymujemy zmienne, które mogą mieć wartość null

```
var unknown:String? = null
println(unknown)
unknown="Test"
println(unknown)
```

```
null
Test
```

- Zmiennych typu nullowego nie można wprost podstawiać pod zwykłe zmienne

```
val username: String? = null
val firstname: String = username
```

Typy nullowe

- Dodając do nazwy typu ? otrzymujemy zmienne, które mogą mieć wartość null

```
var unknown:String? = null
println(unknown)
unknown="Test"
println(unknown)
```

```
null
Test
```

- Zmiennych typu nullowego nie można wprost podstawiać pod zwykłe zmienne

```
val username: String? = null
val firstname: String = username
```

```
Simplest version.kt
  Warning:(9, 5) Variable 'firstname' is never used
  Error:(9, 25) Type mismatch: inferred type is String? but String was expected
```

Typy nullowe

Typy nullowe

- Aby wykonać takie podstawienie, trzeba najpierw sprawdzić, czy zmienna nie jest pusta

Typy nullowe

- Aby wykonać takie podstawienie, trzeba najpierw sprawdzić, czy zmienna nie jest pusta

```
val username: String? = null

if (username != null) {
     val firstname: String = username
}
```

Typy nullowe

- Aby wykonać takie podstawienie, trzeba najpierw sprawdzić, czy zmienna nie jest pusta

```
val username: String? = null

if (username != null) {
     val firstname: String = username
}
```

- Zmiennej nullowej nie można też wywołać metod ani odwoływać się do jej własności

Typy nullowe

- Aby wykonać takie podstawienie, trzeba najpierw sprawdzić, czy zmienna nie jest pusta

```
val username: String? = null

if (username != null) {
     val firstname: String = username
}
```

- Zmiennej nullowej nie można też wywołać metod ani odwoływać się do jej własności

```
val username: String? = null
val uppercase = username.toUpperCase()
```

Typy nullowe

- Aby wykonać takie podstawienie, trzeba najpierw sprawdzić, czy zmienna nie jest pusta

```
val username: String? = null

if (username != null) {
     val firstname: String = username
}
```

- Zmiennej nullowej nie można też wywołać metod ani odwoływać się do jej własności

```
val username: String? = null
val uppercase = username.toUpperCase()
```

```
Simplest version.kt
    Warning:(9, 8) Variable 'uppercase' is never used
    Error:(9, 28) Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?
```

Typy nullowe

Typy nullowe

- Można użyć skróconej instrukcji

Typy nullowe

- Można użyć skróconej instrukcji

```
val username: String? = null
if (username != null) {
     val uppercase = username.toUpperCase()
}
```

Typy nullowe

- Można użyć skróconej instrukcji

```
val username: String? = null
if (username != null) {
     val uppercase = username.toUpperCase()
}
```

```
val username: String? = null
val uppercase = username?.toUpperCase()
val length = username?.length

val i:Int = length
```

Typy nullowe

- Można użyć skróconej instrukcji

```
val username: String? = null
if (username != null) {
     val uppercase = username.toUpperCase()
}
```

```
val username: String? = null
val uppercase = username?.toUpperCase()
val length = username?.length

val i:Int = length
```

- A także wymusić konwersję na typ nienullowy

Typy nullowe

- Można użyć skróconej instrukcji

```
val username: String? = null
if (username != null) {
     val uppercase = username.toUpperCase()
}
```

```
val username: String? = null
val uppercase = username?.toUpperCase()
val length = username?.length

val i:Int = length
```

- A także wymusić konwersję na typ nienullowy

```
val username: String? = null
val length = username!!.length
```

Typy nullowe

- Można użyć skróconej instrukcji

```
val username: String? = null
if (username != null) {
     val uppercase = username.toUpperCase()
}
```

```
val username: String? = null
val uppercase = username?.toUpperCase()
val length = username?.length

val i:Int = length
```

- A także wymusić konwersję na typ nienullowy

```
val username: String? = null
val length = username!!.length
```

```
Exception in thread "main" kotlin.KotlinNullPointerException
    at Simplest_versionKt.main(Simplest_version.kt:9)
```

Typy nullowe

Typy nullowe

- A co, jak zmienną nullową chcemy przekazać jako argument funkcji?

Typy nullowe

- A co, jak zmienną nullową chcemy przekazać jako argument funkcji?

```
val firstNumber = 10
val secondNumber: Int? = 20

val result = firstNumber.times(secondNumber)
print(result)
```

Typy nullowe

- A co, jak zmienną nullową chcemy przekazać jako argument funkcji?

```
val firstNumber = 10
val secondNumber: Int? = 20

val result = firstNumber.times(secondNumber)
print(result)
```

```
Simplest version.kt
    Error:(11, 32) Type mismatch: inferred type is Int? but Int was expected
```

Typy nullowe

- A co, jak zmienną nullową chcemy przekazać jako argument funkcji?

```
val firstNumber = 10
val secondNumber: Int? = 20

val result = firstNumber.times(secondNumber)
print(result)
```

```
Simplest version.kt
    Error:(11, 32) Type mismatch: inferred type is Int? but Int was expected
```

- Można użyć funkcji **let**

Typy nullowe

- A co, jak zmienną nullową chcemy przekazać jako argument funkcji?

```
val firstNumber = 10
val secondNumber: Int? = 20

val result = firstNumber.times(secondNumber)
print(result)
```

```
Simplest version.kt
    Error:(11, 32) Type mismatch: inferred type is Int? but Int was expected
```

- Można użyć funkcji **let**

```
val firstNumber = 10
val secondNumber: Int? = 20

secondNumber?.let {
    val result = firstNumber.times(it)
print(result)
```

Typy nullowe

- A co, jak zmienną nullową chcemy przekazać jako argument funkcji?

```
val firstNumber = 10
val secondNumber: Int? = 20

val result = firstNumber.times(secondNumber)
print(result)
```

```
Simplest version.kt
    Error:(11, 32) Type mismatch: inferred type is Int? but Int was expected
```

- Można użyć funkcji **let**

```
val firstNumber = 10
val secondNumber: Int? = 20

secondNumber?.let {
    val result = firstNumber.times(it)
print(result)
```

```
Compilation completed successfully
200
```

Typy nullowe

Typy nullowe

- Można też skorzystać z operatora ?:

Typy nullowe

- Można też skorzystać z operatora ?:

```
val myString:String? = null

if (myString != null) {
   return myString
} else {
    return "String is null"
}
```

Typy nullowe

- Można też skorzystać z operatora ?:

```
val myString:String? = null

if (myString != null) {
   return myString
} else {
    return "String is null"
}
```

```
val myString:String? = null

return myString ?: "String is null"
```

Operator rzutowania

Operator rzutowania

- Operator as

Operator rzutowania

- Operator as
  - Wersja niebezpieczna

Operator rzutowania

- Operator as
  - Wersja niebezpieczna

```
val keyMgr = getSystemService(Context.KEYGUARD_SERVICE) as KeyguardManager
```

Operator rzutowania

- Operator as
  - Wersja niebezpieczna

```
val keyMgr = getSystemService(Context.KEYGUARD_SERVICE) as KeyguardManager
```

  - Wersja bezpieczna

Operator rzutowania

- Operator as
  - Wersja niebezpieczna

```
val keyMgr = getSystemService(Context.KEYGUARD_SERVICE) as KeyguardManager
```

  - Wersja bezpieczna

```
val keyMgr = getSystemService(Context.KEYGUARD_SERVICE) as? KeyguardManager
```

Operator rzutowania

- Operator as
  - Wersja niebezpieczna

```
val keyMgr = getSystemService(Context.KEYGUARD_SERVICE) as KeyguardManager
```

  - Wersja bezpieczna

```
val keyMgr = getSystemService(Context.KEYGUARD_SERVICE) as? KeyguardManager
```

- Operator is

Operator rzutowania

- Operator as
  - Wersja niebezpieczna

```
val keyMgr = getSystemService(Context.KEYGUARD_SERVICE) as KeyguardManager
```

  - Wersja bezpieczna

```
val keyMgr = getSystemService(Context.KEYGUARD_SERVICE) as? KeyguardManager
```

- Operator is

```
if (keyMgr is KeyguardManager) {
 // It is a KeyguardManager object
}
```

Operatory

Operatory

-	Negacja
+, -, *, /	Arytmetyczne
%	Modulo
+=, -=, *=, /=, %=	Skrócone
++, - -	Inkrementacji, dekrementacji (postfix i prefix)
==, !=	Porównania
>, <, >=,<=	Porównania
!	Zaprzeczenie
&&, ||	And i Or
..	Operator zakresu

Pętle

Pętle

- Pętla for-in

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

```
for (index in "Hello") {
    println("Value of index is $index")
}
```

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

```
for (index in "Hello") {
    println("Value of index is $index")
}
```

```
Value of index is H
Value of index is e
Value of index is l
Value of index is l
Value of index is o
```

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

```
for (index in "Hello") {
    println("Value of index is $index")
}
```

```
Value of index is H
Value of index is e
Value of index is l
Value of index is l
Value of index is o
```

  - W dół (downTo)

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

```
for (index in "Hello") {
    println("Value of index is $index")
}
```

```
Value of index is H
Value of index is e
Value of index is l
Value of index is l
Value of index is o
```

  - W dół (downTo)

```
for (index in 100 downTo 90) {
    print("$index.. ")
}
```

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

```
for (index in "Hello") {
    println("Value of index is $index")
}
```

```
Value of index is H
Value of index is e
Value of index is l
Value of index is l
Value of index is o
```

  - W dół (downTo)

```
for (index in 100 downTo 90) {
    print("$index.. ")
}
```

```
100.. 99.. 98.. 97.. 96.. 95.. 94.. 93.. 92.. 91.. 90..
```

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

```
for (index in "Hello") {
    println("Value of index is $index")
}
```

```
Value of index is H
Value of index is e
Value of index is l
Value of index is l
Value of index is o
```

  - W dół (downTo)

```
for (index in 100 downTo 90) {
    print("$index.. ")
}
```

```
100.. 99.. 98.. 97.. 96.. 95.. 94.. 93.. 92.. 91.. 90..
```

  - I w górę (until)

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

```
for (index in "Hello") {
    println("Value of index is $index")
}
```

```
Value of index is H
Value of index is e
Value of index is l
Value of index is l
Value of index is o
```

  - W dół (downTo)

```
for (index in 100 downTo 90) {
    print("$index.. ")
}
```

```
100.. 99.. 98.. 97.. 96.. 95.. 94.. 93.. 92.. 91.. 90..
```

  - I w górę (until)

```
for (index in 1 until 10) {
    print("$index.. ")
}
```

Pętle

- Pętla for-in
  - Z operatorem zakresu ..

```
for (index in 1..5) {
    println("Value of index is $index")
}
```

```
Value of index is 1
Value of index is 2
Value of index is 3
Value of index is 4
Value of index is 5
```

  - Dla kolekcji (akurat String)

```
for (index in "Hello") {
    println("Value of index is $index")
}
```

```
Value of index is H
Value of index is e
Value of index is l
Value of index is l
Value of index is o
```

  - W dół (downTo)

```
for (index in 100 downTo 90) {
    print("$index.. ")
}
```

```
100.. 99.. 98.. 97.. 96.. 95.. 94.. 93.. 92.. 91.. 90..
```

  - I w górę (until)

```
for (index in 1 until 10) {
    print("$index.. ")
}
```

```
1.. 2.. 3.. 4.. 5.. 6.. 7.. 8.. 9..
```

Pętle

Pętle

- Pętla for-in

Pętle

- Pętla for-in
  - Z krokiem step

Pętle

- Pętla for-in
  - Z krokiem step

```
for (index in 0 until 100 step 10) {
    print("$index.. ")
}
```

Pętle

- Pętla for-in
  - Z krokiem step

```
for (index in 0 until 100 step 10) {
    print("$index.. ")
}
```

```
0.. 10.. 20.. 30.. 40.. 50.. 60.. 70.. 80.. 90..
```

Pętle

- Pętla for-in
  - Z krokiem step

```
for (index in 0 until 100 step 10) {
    print("$index.. ")
}
```

```
0.. 10.. 20.. 30.. 40.. 50.. 60.. 70.. 80.. 90..
```

- Pętla while

Pętle

- Pętla for-in
  - Z krokiem step

```
for (index in 0 until 100 step 10) {
    print("$index.. ")
}
```

```
0.. 10.. 20.. 30.. 40.. 50.. 60.. 70.. 80.. 90..
```

  - Pętla while

```
var myCount = 0
while  (myCount < 10) {
    myCount++
    println(myCount)
}
```

Pętle

- Pętla for-in
  - Z krokiem step

```
for (index in 0 until 100 step 10) {
    print("$index.. ")
}
```

```
0.. 10.. 20.. 30.. 40.. 50.. 60.. 70.. 80.. 90..
```

- Pętla while

```
var myCount = 0
while  (myCount < 10) {
    myCount++
    println(myCount)
}
```

```
1
2
3
4
5
6
7
8
9
10
```

Pętle

- Pętla for-in
  - Z krokiem step

```
for (index in 0 until 100 step 10) {
    print("$index.. ")
}
```

```
0.. 10.. 20.. 30.. 40.. 50.. 60.. 70.. 80.. 90..
```

- Pętla while

```
var myCount = 0
while  (myCount < 10) {
    myCount++
    println(myCount)
}
```

```
1
2
3
4
5
6
7
8
9
10
```

- Pętla do-while

Pętle

- Pętla for-in
  - Z krokiem step

```
for (index in 0 until 100 step 10) {
    print("$index.. ")
}
```

```
0.. 10.. 20.. 30.. 40.. 50.. 60.. 70.. 80.. 90..
```

- Pętla while

```
var myCount = 0
while  (myCount < 10) {
    myCount++
    println(myCount)
}
```

```
1
2
3
4
5
6
7
8
9
10
```

- Pętla do-while

```
var i = 5
do {
    i--
    println(i)
} while (i > 0)
```

Pętle

- Pętla for-in
  - Z krokiem step

```
for (index in 0 until 100 step 10) {
    print("$index.. ")
}
```

```
0.. 10.. 20.. 30.. 40.. 50.. 60.. 70.. 80.. 90..
```

- Pętla while

```
var myCount = 0
while  (myCount < 10) {
    myCount++
    println(myCount)
}
```

```
1
2
3
4
5
6
7
8
9
10
```

- Pętla do-while

```
var i = 5
do {
    i--
    println(i)
} while (i > 0)
```

```
4
3
2
1
0
```

Pętle

Pętle

- Przerwanie pętli - break

Pętle

- Przerwanie pętli - break

```
var j = 10
for (i in 0..100)
{
    j += j
    if (j > 100) {
        break
    }
    println("j = $j")
}
```

Pętle

- Przerwanie pętli - break

```
var j = 10
for (i in 0..100)
{
    j += j
    if (j > 100) {
        break
    }
    println("j = $j")
}
```

```
j = 20
j = 40
j = 80
```

Pętle

- Przerwanie pętli - break

```
var j = 10
for (i in 0..100)
{
    j += j
    if (j > 100) {
        break
    }
    println("j = $j")
}
```

```
j = 20
j = 40
j = 80
```

- Kontynuowanie wykonania pętli - continue

Pętle

- Przerwanie pętli - break

```
var j = 10
for (i in 0..100)
{
    j += j
    if (j > 100) {
        break
    }
    println("j = $j")
}
```

```
j = 20
j = 40
j = 80
```

- Kontynuowanie wykonania pętli - continue

```
var i = 1
while (i < 20)
{
    i += 1
    if (i % 2 != 0) {
        continue
    }
     println("i = $i")
}
```

Pętle

- Przerwanie pętli - break

```
var j = 10
for (i in 0..100)
{
    j += j
    if (j > 100) {
        break
    }
    println("j = $j")
}
```

```
j = 20
j = 40
j = 80
```

- Kontynuowanie wykonania pętli - continue

```
var i = 1
while (i < 20)
{
    i += 1
    if (i % 2 != 0) {
        continue
    }
     println("i = $i")
}
```

```
i = 2
i = 4
i = 6
i = 8
i = 10
i = 12
i = 14
i = 16
i = 18
i = 20
```

Pętle

Pętle

- Pętle zagnieżdżone z etykietami

Pętle

- Pętle zagnieżdżone z etykietami

```
outerloop@ for (i in 1..100) {
    println("Outer loop i = $i")

    for (j in 1..100) {
       println("Inner loop j = $j")
       if (j == 10) break@outerloop
    }
}
```

Pętle

- Pętle zagnieżdżone z etykietami

```
outerloop@ for (i in 1..100) {
    println("Outer loop i = $i")

    for (j in 1..100) {
       println("Inner loop j = $j")
       if (j == 10) break@outerloop
    }
}
```

```
Outer loop i = 1
Inner loop j = 1
Inner loop j = 2
Inner loop j = 3
Inner loop j = 4
Inner loop j = 5
Inner loop j = 6
Inner loop j = 7
Inner loop j = 8
Inner loop j = 9
Inner loop j = 10
```

Pętle

- Pętle zagnieżdżone z etykietami

```
outerloop@ for (i in 1..100) {
    println("Outer loop i = $i")

    for (j in 1..100) {
       println("Inner loop j = $j")
       if (j == 10) break@outerloop
    }
}
```

```
Outer loop i = 1
Inner loop j = 1
Inner loop j = 2
Inner loop j = 3
Inner loop j = 4
Inner loop j = 5
Inner loop j = 6
Inner loop j = 7
Inner loop j = 8
Inner loop j = 9
Inner loop j = 10
```

```
outerloop@ for (i in 1..100) {
    println("Outer loop i = $i")

    for (j in 1..100) {
       println("Inner loop j = $j")
       if (j == 5) continue@outerloop
    }
}
```

Pętle

- Pętle zagnieżdżone z etykietami

```
outerloop@ for (i in 1..100) {
    println("Outer loop i = $i")

    for (j in 1..100) {
       println("Inner loop j = $j")
       if (j == 10) break@outerloop
    }
}
```

```
Outer loop i = 1
Inner loop j = 1
Inner loop j = 2
Inner loop j = 3
Inner loop j = 4
Inner loop j = 5
Inner loop j = 6
Inner loop j = 7
Inner loop j = 8
Inner loop j = 9
Inner loop j = 10
```

```
outerloop@ for (i in 1..100) {
    println("Outer loop i = $i")

    for (j in 1..100) {
       println("Inner loop j = $j")
       if (j == 5) continue@outerloop
    }
}
```

```
Outer loop i = 1
Inner loop j = 1
Inner loop j = 2
Inner loop j = 3
Inner loop j = 4
Inner loop j = 5
Outer loop i = 2
Inner loop j = 1
Inner loop j = 2
Inner loop j = 3
Inner loop j = 4
Inner loop j = 5
```

Instrukcje warunkowe

Instrukcje warunkowe

- Instrukcja if - jak w większości języków

Instrukcje warunkowe

- Instrukcja if - jak w większości języków

```
val x=5; val y=7

val largest = if (x > y) "x is greatest" else "y is greatest"
println(largest)
```

Instrukcje warunkowe

- Instrukcja if - jak w większości języków

```
val x=5; val y=7

val largest = if (x > y) "x is greatest" else "y is greatest"
println(largest)
```

```
y is greatest
```

Instrukcje warunkowe

- Instrukcja if - jak w większości języków

```
val x=5; val y=7

val largest = if (x > y) "x is greatest" else "y is greatest"
println(largest)
```

```
y is greatest
```

- Instrukcja when - odpowiednik switch

Instrukcje warunkowe

- Instrukcja if - jak w większości języków

```
val x=5; val y=7

val largest = if (x > y) "x is greatest" else "y is greatest"
println(largest)
```

```
y is greatest
```

- Instrukcja when - odpowiednik switch

```
when (x) {
    10 -> println ("x is 10")
    9 -> println("x is 9")
    8 -> println("x is 8")
    else ->  println("x is less than 8")
}
```

Instrukcje warunkowe

- Instrukcja if - jak w większości języków

```
val x=5; val y=7

val largest = if (x > y) "x is greatest" else "y is greatest"
println(largest)
```

```
y is greatest
```

- Instrukcja when - odpowiednik switch

```
when (x) {
    10 -> println ("x is 10")
    9 -> println("x is 9")
    8 -> println("x is 8")
    else ->  println("x is less than 8")
}
```

```
x is less than 8
```

Funkcje

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

```
fun buildMessageFor(name: String, count: Int): String {
        return("$name, you are customer number $count")
}
```

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

```
fun buildMessageFor(name: String, count: Int): String {
        return("$name, you are customer number $count")
}
```

- Wywołanie funkcji

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

```
fun buildMessageFor(name: String, count: Int): String {
        return("$name, you are customer number $count")
}
```

- Wywołanie funkcji

```
sayHello()
buildMessageFor("John", 10)
```

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

```
fun buildMessageFor(name: String, count: Int): String {
        return("$name, you are customer number $count")
}
```

- Wywołanie funkcji

```
sayHello()
buildMessageFor("John", 10)
```

```
Hello
John, you are customer number 10
```

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

```
fun buildMessageFor(name: String, count: Int): String {
        return("$name, you are customer number $count")
}
```

- Wywołanie funkcji

```
sayHello()
buildMessageFor("John", 10)
```

```
Hello
John, you are customer number 10
```

- Funkcje jednoliniowe

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

```
fun buildMessageFor(name: String, count: Int): String {
        return("$name, you are customer number $count")
}
```

- Wywołanie funkcji

```
sayHello()
buildMessageFor("John", 10)
```

```
Hello
John, you are customer number 10
```

- Funkcje jednoliniowe

```
fun multiply(x: Int, y: Int): Int = x * y
```

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

```
fun buildMessageFor(name: String, count: Int): String {
        return("$name, you are customer number $count")
}
```

- Wywołanie funkcji

```
sayHello()
buildMessageFor("John", 10)
```

```
Hello
John, you are customer number 10
```

- Funkcje jednoliniowe

```
fun multiply(x: Int, y: Int): Int = x * y
```

lub z inferencją

Funkcje

- Deklarowane z użyciem słowa kluczowego **fun**
  - Bez typu

```
fun sayHello() {
    println("Hello")
}
```

  - Z typem

```
fun buildMessageFor(name: String, count: Int): String {
        return("$name, you are customer number $count")
}
```

- Wywołanie funkcji

```
sayHello()
buildMessageFor("John", 10)
```

```
Hello
John, you are customer number 10
```

- Funkcje jednoliniowe

```
fun multiply(x: Int, y: Int): Int = x * y
```

lub z inferencją

```
fun multiply(x: Int, y: Int) = x * y
```

Funkcje

Funkcje

- Funkcje zagnieżdżone

Funkcje

- Funkcje zagnieżdżone

```
fun main(args: Array<String>) {

    val name = "John"
    val count = 5

    fun displayString() {
        for (index in 0..count) {
            println(name)
        }
    }
    displayString()
}
```

Funkcje

- Funkcje zagnieżdżone

```
fun main(args: Array<String>) {

    val name = "John"
    val count = 5

    fun displayString() {
        for (index in 0..count) {
            println(name)
        }
    }
    displayString()
}
```

```
John
John
John
John
John
John
```

Funkcje

- Funkcje zagnieżdżone

```
fun main(args: Array<String>) {

    val name = "John"
    val count = 5

    fun displayString() {
        for (index in 0..count) {
            println(name)
        }
    }
    displayString()
}
```

```
John
John
John
John
John
John
```

- Mają dostęp do zmienny funkcji wyższego poziomu

Funkcje

Funkcje

- Domyślne argumenty

Funkcje

- Domyślne argumenty

```
fun buildMessageFor(name: String = "Customer", count: Int = 0): String {
        return("$name, you are customer number $count")
}
```

Funkcje

- Domyślne argumenty

```
fun buildMessageFor(name: String = "Customer", count: Int = 0): String {
        return("$name, you are customer number $count")
}
```

```
val message = buildMessageFor(count = 10)
println(message)
```

Funkcje

- Domyślne argumenty

```
fun buildMessageFor(name: String = "Customer", count: Int = 0): String {
        return("$name, you are customer number $count")
}
```

```
val message = buildMessageFor(count = 10)
println(message)
```

```
Customer, you are customer number 10
```

Funkcje

- Domyślne argumenty

```
fun buildMessageFor(name: String = "Customer", count: Int = 0): String {
        return("$name, you are customer number $count")
}
```

```
val message = buildMessageFor(count = 10)
println(message)
```

```
Customer, you are customer number 10
```

- Zmienna liczba argumentów

Funkcje

- Domyślne argumenty

```
fun buildMessageFor(name: String = "Customer", count: Int = 0): String {
        return("$name, you are customer number $count")
}
```

```
val message = buildMessageFor(count = 10)
println(message)
```

```
Customer, you are customer number 10
```

- Zmienna liczba argumentów

```
fun displayStrings(vararg strings: String)
{
    for (string in strings) {
        println(string)
    }
}
```

Funkcje

- Domyślne argumenty

```
fun buildMessageFor(name: String = "Customer", count: Int = 0): String {
        return("$name, you are customer number $count")
}
```

```
val message = buildMessageFor(count = 10)
println(message)
```

```
Customer, you are customer number 10
```

- Zmienna liczba argumentów

```
fun displayStrings(vararg strings: String)
{
    for (string in strings) {
        println(string)
    }
}
```

```
displayStrings("one", "two", "three", "four")
```

Funkcje

- Domyślne argumenty

```
fun buildMessageFor(name: String = "Customer", count: Int = 0): String {
        return("$name, you are customer number $count")
}
```

```
val message = buildMessageFor(count = 10)
println(message)
```

```
Customer, you are customer number 10
```

- Zmienna liczba argumentów

```
fun displayStrings(vararg strings: String)
{
    for (string in strings) {
        println(string)
    }
}
```

```
displayStrings("one", "two", "three", "four")
```

```
one
two
three
four
```

Wyrażenia Lambda

Wyrażenia Lambda

- Są to zamknięte fragmenty kodu, podobnie jak funkcje

Wyrażenia Lambda

- Są to zamknięte fragmenty kodu, podobnie jak funkcje

```
val sayHello = { println("Hello") }
sayHello()
```

Wyrażenia Lambda

- Są to zamknięte fragmenty kodu, podobnie jak funkcje

```
val sayHello = { println("Hello") }
sayHello()
```

```
val multiply = {val1: Int, val2: Int -> val1 * val2 }
val result = multiply(10, 20)
```

Wyrażenia Lambda

- Są to zamknięte fragmenty kodu, podobnie jak funkcje

```
val sayHello = { println("Hello") }
sayHello()
```

```
val multiply = {val1: Int, val2: Int -> val1 * val2 }
val result = multiply(10, 20)
```

```
val result = {val1: Int, val2: Int -> val1 * val2 }(10, 20)
```

Wyrażenia Lambda

- Są to zamknięte fragmenty kodu, podobnie jak funkcje

```
val sayHello = { println("Hello") }
sayHello()
```

```
val multiply = {val1: Int, val2: Int -> val1 * val2 }
val result = multiply(10, 20)
```

```
val result = {val1: Int, val2: Int -> val1 * val2 }(10, 20)
```

- Można je przekazywać jako argumenty

Wyrażenia Lambda

- Są to zamknięte fragmenty kodu, podobnie jak funkcje

```
val sayHello = { println("Hello") }
sayHello()
```

```
val multiply = {val1: Int, val2: Int -> val1 * val2 }
val result = multiply(10, 20)
```

```
val result = {val1: Int, val2: Int -> val1 * val2 }(10, 20)
```

- Można je przekazywać jako argumenty

```
fun inchesToFeet (inches: Double): Double {
    return inches * 0.0833333
}

fun inchesToYards (inches: Double): Double {
    return inches * 0.0277778
}

fun outputConversion(converterFunc: (Double) -> Double, value: Double) {
    val result = converterFunc(value)
    println("Result of conversion is $result")
}
```

Wyrażenia Lambda

- Są to zamknięte fragmenty kodu, podobnie jak funkcje

```
val sayHello = { println("Hello") }
sayHello()
```

```
val multiply = {val1: Int, val2: Int -> val1 * val2 }
val result = multiply(10, 20)
```

```
val result = {val1: Int, val2: Int -> val1 * val2 }(10, 20)
```

- Można je przekazywać jako argumenty

```
fun inchesToFeet (inches: Double): Double {
    return inches * 0.0833333
}

fun inchesToYards (inches: Double): Double {
    return inches * 0.0277778
}

fun outputConversion(converterFunc: (Double) -> Double, value: Double) {
    val result = converterFunc(value)
    println("Result of conversion is $result")
}
```

```
outputConversion(::inchesToFeet, 22.45)
outputConversion(::inchesToYards, 22.45)
```

Wyrażenia Lambda

- Są to zamknięte fragmenty kodu, podobnie jak funkcje

```
val sayHello = { println("Hello") }
sayHello()
```

```
val multiply = {val1: Int, val2: Int -> val1 * val2 }
val result = multiply(10, 20)
```

```
val result = {val1: Int, val2: Int -> val1 * val2 }(10, 20)
```

- Można je przekazywać jako argumenty

```
fun inchesToFeet (inches: Double): Double {
    return inches * 0.0833333
}

fun inchesToYards (inches: Double): Double {
    return inches * 0.0277778
}

fun outputConversion(converterFunc: (Double) -> Double, value: Double) {
    val result = converterFunc(value)
    println("Result of conversion is $result")
}
```

```
outputConversion(::inchesToFeet, 22.45)
outputConversion(::inchesToYards, 22.45)
```

```
Result of conversion is 1.8708325849999998
Result of conversion is 0.62361161
```

Klasy

Klasy

- Do deklaracji klasy mamy słowo kluczowe class

Klasy

- Do deklaracji klasy mamy słowo kluczowe class

```
class BankAccount {

}
```

Klasy

- Do deklaracji klasy mamy słowo kluczowe class

```
class BankAccount {

}
```

- Klasa z własnościami

Klasy

- Do deklaracji klasy mamy słowo kluczowe class

```
class BankAccount {

}
```

- Klasa z własnościami

```
class BankAccount {
    var accountBalance: Double = 0.0
    var accountNumber: Int = 0
}
```

Klasy

- Do deklaracji klasy mamy słowo kluczowe class

```
class BankAccount {

}
```

- Klasa z własnościami

```
class BankAccount {
    var accountBalance: Double = 0.0
    var accountNumber: Int = 0
}
```

- Klasa z metodą

Klasy

- Do deklaracji klasy mamy słowo kluczowe class

```
class BankAccount {

}
```

- Klasa z własnościami

```
class BankAccount {
    var accountBalance: Double = 0.0
    var accountNumber: Int = 0
}
```

- Klasa z metodą

```
class BankAccount {
    var accountBalance: Double = 0.0
    var accountNumber: Int = 0

    fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

Klasy

Klasy

- Utworzenie obiektu

Klasy

- Utworzenie obiektu

```
val account1: BankAccount = BankAccount()
```

Klasy

- Utworzenie obiektu

```
val account1: BankAccount = BankAccount()
```

lub

Klasy

- Utworzenie obiektu

```
val account1: BankAccount = BankAccount()
```

lub

```
val account1 = BankAccount()
```

Klasy

- Utworzenie obiektu

```
val account1: BankAccount = BankAccount()
```

lub

```
val account1 = BankAccount()
```

- Dodanie konstruktora drugiego rzędu

Klasy

- Utworzenie obiektu

```
val account1: BankAccount = BankAccount()
```

lub

```
val account1 = BankAccount()
```

- Dodanie konstruktora drugiego rzędu

```
class BankAccount {
    var accountBalance: Double = 0.0
    var accountNumber: Int = 0

    constructor(number: Int, balance: Double) {
        accountNumber =  number
        accountBalance = balance
    }

    fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

Klasy

Klasy

- Konstruktorów może oczywiście być wiele

Klasy

- Konstruktorów może oczywiście być wiele

```
class BankAccount {

    var accountBalance: Double = 0.0
    var accountNumber: Int = 0
    var lastName: String = ""

    constructor(number: Int,
                balance: Double) {
        accountNumber =  number
        accountBalance = balance
    }

    constructor(number: Int,
                balance: Double,
               name: String ) {
        accountNumber =  number
        accountBalance = balance
        lastName = name
    }

    fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

Klasy

Klasy

- Konstruktor pierwszego rzędu można zdefiniować w deklaracji klasy

Klasy

- Konstruktor pierwszego rzędu można zdefiniować w deklaracji klasy

```
fun main(args: Array<String>) {

    val account1 = BankAccount(1234,10.0)
    account1.displayBalance()
}

class BankAccount (val accountNumber: Int, var accountBalance: Double) {

    var lastName: String = ""

    constructor(accountNumber: Int,
                accountBalance: Double,
                name: String ) : this(accountNumber, accountBalance) {

        lastName = name
    }

    fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

Klasy

- Konstruktor pierwszego rzędu można zdefiniować w deklaracji klasy

```
fun main(args: Array<String>) {

    val account1 = BankAccount(1234,10.0)
    account1.displayBalance()
}

class BankAccount (val accountNumber: Int, var accountBalance: Double) {

    var lastName: String = ""

    constructor(accountNumber: Int,
                accountBalance: Double,
                name: String ) : this(accountNumber, accountBalance) {

        lastName = name
    }

    fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

```
Number 1234
Current balance is 10.0
```

Klasy

Klasy

- Oprócz konstruktorów klasy mogą też posiadać inicjalizator, przydatny przy wywołaniu konstruktorów pierwszego rzędu, które nie mogą zawierać własnego kodu

Klasy

- Oprócz konstruktorów klasy mogą też posiadać inicjalizator, przydatny przy wywołaniu konstruktorów pierwszego rzędu, które nie mogą zawierać własnego kodu

```
class BankAccount (val accountNumber: Int, var accountBalance: Double) {

    var lastName: String = ""

    init {
       lastName = "Smith"
    }

    fun displayBalance()
    {
       println("Last name $lastName")
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

Klasy

- Oprócz konstruktorów klasy mogą też posiadać inicjalizator, przydatny przy wywołaniu konstruktorów pierwszego rzędu, które nie mogą zawierać własnego kodu

```
class BankAccount (val accountNumber: Int, var accountBalance: Double) {

    var lastName: String = ""

    init {
        lastName = "Smith"
    }

    fun displayBalance()
    {
        println("Last name $lastName")
        println("Number $accountNumber")
        println("Current balance is $accountBalance")
    }
}
```

```
Last name Smith
Number 1234
Current balance is 10.0
```

Klasy

Klasy

- Dostosowany dostęp do własności - można zdefiniować własny getter i setter

Klasy

- Dostosowany dostęp do własności - można zdefiniować własny getter i setter

```
class BankAccount (val accountNumber: Int, var accountBalance: Double) {

    val fees: Double = 25.00

    val balanceLessFees: Double
        get() {
            return accountBalance - fees
        }

    fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

Klasy

- Dostosowany dostęp do własności - można zdefiniować własny getter i setter

```
class BankAccount (val accountNumber: Int, var accou

    val fees: Double = 25.00

    val balanceLessFees: Double
        get() {
            return accountBalance - fees
        }

    fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

```
class BankAccount (val accountNumber: Int, var accountBalance: Double) {

    val fees: Double = 25.00

    var balanceLessFees: Double
        get() {
            return accountBalance - fees
        }
        set(value) {
            accountBalance = value - fees
        }

    fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}
```

Klasy

Klasy

- Klasy zagnieżdżone

Klasy

- Klasy zagnieżdżone
  - Klasa wewnętrzna nie ma dostępu do klasy zewnętrznej

Klasy

- Klasy zagnieżdżone
  - Klasa wewnętrzna nie ma dostępu do klasy zewnętrznej

```
class ClassA {
    class ClassB {
    }
}
```

Klasy

- Klasy zagnieżdżone
  - Klasa wewnętrzna nie ma dostępu do klasy zewnętrznej

```
class ClassA {
    class ClassB {
    }
}
```

  - Klasa wewnętrzna ma teraz dostęp do klasy zewnętrznej - inner

Klasy

- Klasy zagnieżdżone
  - Klasa wewnętrzna nie ma dostępu do klasy zewnętrznej

```
class ClassA {
    class ClassB {
    }
}
```

  - Klasa wewnętrzna ma teraz dostęp do klasy zewnętrznej - inner

```
class ClassA {
        var myProperty: Int = 10

    inner class ClassB {
            val result = 20 + myProperty
    }
}
```

Dziedziczenie

Dziedziczenie

- Dziedziczyć można tylko po pojedynczej klasie

Dziedziczenie

- Dziedziczyć można tylko po pojedynczej klasie
- Aby móc dziedziczyć po klasie musi być ona zadeklarowana jako otwarta - open

Dziedziczenie

- Dziedziczyć można tylko po pojedynczej klasie
- Aby móc dziedziczyć po klasie musi być ona zadeklarowana jako otwarta - open

```
open class MyParentClass {
    var myProperty: Int = 0
}

class MySubClass : MyParentClass() {

}
```

Dziedziczenie

- Dziedziczyć można tylko po pojedynczej klasie
- Aby móc dziedziczyć po klasie musi być ona zadeklarowana jako otwarta - open

```
open class MyParentClass {
    var myProperty: Int = 0
}

class MySubClass : MyParentClass() {

}
```

- Gdy klasa nadrzędna ma konstruktor pierwszego rzędu sprawa się trochę komplikuje

Dziedziczenie

- Dziedziczyć można tylko po pojedynczej klasie
- Aby móc dziedziczyć po klasie musi być ona zadeklarowana jako otwarta - open

```
open class MyParentClass {
    var myProperty: Int = 0
}

class MySubClass : MyParentClass() {

}
```

- Gdy klasa nadrzędna ma konstruktor pierwszego rzędu sprawa się trochę komplikuje

```
open class MyParentClass(var myProperty: Int) {

}

class MySubClass(myProperty: Int) : MyParentClass(myProperty) {

}
```

Dziedziczenie

Dziedziczenie

- Klasa dziedzicząca musi przeciążyć wszystkie konstruktory drugiego rzędu klasy nadrzędnej

Dziedziczenie

- Klasa dziedzicząca musi przeciążyć wszystkie konstruktory drugiego rzędu klasy nadrzędnej

```
open class MyParentClass {
     var myProperty: Int = 0

     constructor(number: Int) {
         myProperty = number
     }
}

class MySubClass : MyParentClass {

    constructor(number: Int) : super(number) {
        // Subclass constructor code here
    }
}
```

Dziedziczenie

Dziedziczenie

- Przykład

Dziedziczenie

- Przykład

```
open class BankAccount {

    var accountNumber = 0
    var accountBalance = 0.0

    constructor(number: Int, balance: Double) {
        accountNumber = number
        accountBalance = balance
    }

    open fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
    }
}

class SavingsAccount : BankAccount {
    var interestRate: Double = 0.0

    constructor(accountNumber: Int, accountBalance: Double) :
                            super(accountNumber, accountBalance)

    fun calculateInterest(): Double
    {
        return interestRate * accountBalance
    }
}
```

Dziedziczenie

Dziedziczenie

- Przeciążanie metod - override - metoda też musi być zadeklarowana jako otwarta - open

Dziedziczenie

- Przeciążanie metod - override - metoda też musi być zadeklarowana jako otwarta - open

```
class SavingsAccount : BankAccount {
    var interestRate: Double = 0.0

    constructor(accountNumber: Int, accountBalance: Double) :
        super(accountNumber, accountBalance)

    fun calculateInterest(): Double
    {
        return interestRate * accountBalance
    }

    override fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
       println("Prevailing interest rate is $interestRate")
    }
}
```

Dziedziczenie

- Przeciążanie metod - override - metoda też musi być zadeklarowana jako otwarta - open

```
class SavingsAccount : BankAccount {
    var interestRate: Double = 0.0

    constructor(accountNumber: Int, accountBalance: Double) :
        super(accountNumber, accountBalance)

    fun calculateInterest(): Double
    {
        return interestRate * accountBalance
    }

    override fun displayBalance()
    {
       println("Number $accountNumber")
       println("Current balance is $accountBalance")
       println("Prevailing interest rate is $interestRate")
    }
}
```

```
override fun displayBalance()
{
    super.displayBalance()
    println("Prevailing interest rate is $interestRate")
}
```

Dziedziczenie

Dziedziczenie

- Dodanie konstruktora drugiego rzędu

Dziedziczenie

- Dodanie konstruktora drugiego rzędu

```
class SavingsAccount : BankAccount {
    var interestRate: Double = 0.0

    constructor(accountNumber: Int, accountBalance: Double) :
        super(accountNumber, accountBalance)

    constructor(accountNumber: Int, accountBalance: Double, rate: Double) :
                            super(accountNumber, accountBalance) {
        interestRate = rate
    }

    fun calculateInterest(): Double
    {
        return interestRate * accountBalance
    }

    override fun displayBalance()
    {
        super.displayBalance()
        println("Prevailing interest rate is $interestRate")
    }
}
```

Dziedziczenie

- Dodanie konstruktora drugiego rzędu

```
class SavingsAccount : BankAccount {
    var interestRate: Double = 0.0

    constructor(accountNumber: Int, accountBalance: Double) :
        super(accountNumber, accountBalance)

    constructor(accountNumber: Int, accountBalance: Double, rate: Double) :
                            super(accountNumber, accountBalance) {
        interestRate = rate
    }

    fun calculateInterest(): Double
    {
        return interestRate * accountBalance
    }

    override fun displayBalance()
    {
        super.displayBalance()
        println("Prevailing interest rate is $interestRate")
    }
}
```

```
val savings1 = SavingsAccount(12311, 600.00, 0.07)
println(savings1.calculateInterest())
savings1.displayBalance()
```

Dziedziczenie

- Dodanie konstruktora drugiego rzędu

```
class SavingsAccount : BankAccount {
    var interestRate: Double = 0.0

    constructor(accountNumber: Int, accountBalance: Double) :
        super(accountNumber, accountBalance)

    constructor(accountNumber: Int, accountBalance: Double, rate: Double) :
                            super(accountNumber, accountBalance) {
        interestRate = rate
    }

    fun calculateInterest(): Double
    {
        return interestRate * accountBalance
    }

    override fun displayBalance()
    {
        super.displayBalance()
        println("Prevailing interest rate is $interestRate")
    }
}
```

```
val savings1 = SavingsAccount(12311, 600.00, 0.07)
println(savings1.calculateInterest())
savings1.displayBalance()
```

```
42.00000000000001
Number 12311
Current balance is 600.0
Prevailing interest rate is 0.07
```

Interfejsy

Interfejsy

- Kotlin oczywiście posiada interfejsy

Interfejsy

- Kotlin oczywiście posiada interfejsy
- Mogą zawierać metody

Interfejsy

- Kotlin oczywiście posiada interfejsy
- Mogą zawierać metody
- Mogą zwierać własności pod warunkiem:

Interfejsy

- Kotlin oczywiście posiada interfejsy
- Mogą zawierać metody
- Mogą zwierać własności pod warunkiem:
  - Są one abstrakcyjne

Interfejsy

- Kotlin oczywiście posiada interfejsy
- Mogą zawierać metody
- Mogą zwierać własności pod warunkiem:
  - Są one abstrakcyjne
  - Mają zdefiniowane funkcje dostępu

Interfejsy

Interfejsy

```
interface MyInterface {
    val prop: Int // abstract

    val propertyWithImplementation: String
        get() = "foo"

    fun foo() {
        print(prop)
    }
}

class Child : MyInterface {
    override val prop: Int = 29
}
```

Typy generyczne

Typy generyczne

- Kotlin ma pełne wsparcie typów generycznych

Typy generyczne

- Kotlin ma pełne wsparcie typów generycznych

```
class Box<T>(t: T) {
    var value = t
}
```

Typy generyczne

- Kotlin ma pełne wsparcie typów generycznych

```
class Box<T>(t: T) {
    var value = t
}
```

```
val box: Box<Int> = Box<Int>(1)
```

Typy generyczne

- Kotlin ma pełne wsparcie typów generycznych

```
class Box<T>(t: T) {
    var value = t
}
```

```
val box: Box<Int> = Box<Int>(1)
```

lub z inferencją typów

Typy generyczne

- Kotlin ma pełne wsparcie typów generycznych

```
class Box<T>(t: T) {
    var value = t
}
```

```
val box: Box<Int> = Box<Int>(1)
```

lub z inferencją typów

```
val box = Box(1)
```

Rozszerzenia

Rozszerzenia

- Kotlin podobnie jak C# lub Swift umożliwia dodawanie rozszerzeń do istniejących typów.

Rozszerzenia

- Kotlin podobnie jak C# lub Swift umożliwia dodawanie rozszerzeń do istniejących typów.

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

Rozszerzenia

- Kotlin podobnie jak C# lub Swift umożliwia dodawanie rozszerzeń do istniejących typów.

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

```
val l = mutableListOf(1, 2, 3)
l.swap(0, 2)
println(l[0])
```

Rozszerzenia

- Kotlin podobnie jak C# lub Swift umożliwia dodawanie rozszerzeń do istniejących typów.

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

```
val l = mutableListOf(1, 2, 3)
l.swap(0, 2)
println(l[0])
```

```
fun <T> MutableList<T>.swap(index1: Int, index2: Int) {
    val tmp = this[index1]
    this[index1] = this[index2]
    this[index2] = tmp
}
```

Rozszerzenia

Rozszerzenia

- Dodać można też własności

Rozszerzenia

- Dodać można też własności

```
val <T> List<T>.lastIndex: Int
    get() = size - 1
```

Rozszerzenia

- Dodać można też własności

```
val <T> List<T>.lastIndex: Int
    get() = size - 1
```

```
val l = mutableListOf(1, 2, 3)
val i = l.lastIndex
println(i)
```